This material is not required and is posted only for the curious. It is a beautiful and powerful proof, but the technical details are not our focus beyond the synergy of preservation and progress. We skipped most of the low-level details in class.

Syntax

\[
\begin{align*}
\text{e} & ::= c \mid \lambda x\ . \ e \mid x \mid e \ e \\
\text{v} & ::= c \mid \lambda x\ . \ e \\
\text{\tau} & ::= \text{int} \mid \tau \rightarrow \tau \\
\Gamma & ::= \cdot \mid \Gamma, x: \tau
\end{align*}
\]

Evaluation Rules (a.k.a. Dynamic Semantics)

\[
\begin{array}{c}
\text{E-Apply} \\
(\lambda x\ . \ e) \ v \rightarrow e[v/x]
\end{array}
\quad
\begin{array}{c}
\text{E-App1} \\
e_1 \rightarrow e'_1 \\
e_1 \ e_2 \rightarrow e'_1 \ e_2
\end{array}
\quad
\begin{array}{c}
\text{E-App2} \\
e_2 \rightarrow e'_2 \\
v \ e_2 \rightarrow v \ e'_2
\end{array}
\]

Typing Rules (a.k.a. Static Semantics)

\[
\begin{array}{c}
\text{T-Const} \\
\Gamma \vdash c : \text{int}
\end{array}
\quad
\begin{array}{c}
\text{T-Var} \\
\Gamma \vdash x : \Gamma(x)
\end{array}
\quad
\begin{array}{c}
\text{T-Fun} \\
\Gamma, x : \tau_1 \vdash e : \tau_2 \\
x \not\in \text{Dom}(\Gamma)
\end{array}
\quad
\begin{array}{c}
\text{T-Fun} \\
\Gamma \vdash \lambda x.\ e : \tau_1 \rightarrow \tau_2
\end{array}
\]

\[
\begin{array}{c}
\text{T-App} \\
\Gamma \vdash e_1 : \tau_2 \rightarrow \tau_1 \\
\Gamma \vdash e_2 : \tau_2
\end{array}
\quad
\begin{array}{c}
\Gamma \vdash e_1 \ e_2 : \tau_1
\end{array}
\]

Type Soundness

Theorem (Type Soundness). If \( \cdot \vdash e : \tau \) and \( e \rightarrow^* e' \), then either \( e' \) is a value or there exists an \( e'' \) such that \( e' \rightarrow e'' \).
Proof

The Type Soundness Theorem follows as a simple corollary to the Progress and Preservation Theorems stated and proven below: Given the Preservation Theorem, a trivial induction on the number of steps taken to reach \( e' \) from \( e \) establishes that \( \cdot \vdash e' : \tau \). Then the Progress Theorem ensures \( e' \) is a value or can step to some \( e'' \).

We need the following lemma for our proof of Progress, below.

**Lemma (Canonical Forms).** If \( \cdot \vdash v : \tau \), then

i. If \( \tau \) is \( \text{int} \), then \( v \) is a constant, i.e., some \( c \).

ii. If \( \tau \) is \( \tau_1 \rightarrow \tau_2 \), then \( v \) is a lambda, i.e., \( \lambda x. e \) for some \( x \) and \( e \).

**Canonical Forms.** The proof is by inspection of the typing rules.

i. If \( \tau \) is \( \text{int} \), then the only rule which lets us give a value this type is T-Const.

ii. If \( \tau \) is \( \tau_1 \rightarrow \tau_2 \), then the only rule which lets us give a value this type is T-Fun.

**Theorem (Progress).** If \( \cdot \vdash e : \tau \), then either \( e \) is a value or there exists some \( e' \) such that \( e \rightarrow e' \).

**Progress.** The proof is by induction on (the height of) the derivation of \( \cdot \vdash e : \tau \), proceeding by cases on the bottommost rule used in the derivation.

T-Const \( e \) is a constant, which is a value, so we are done.

T-Var Impossible, as \( \Gamma \) is \( \cdot \).

T-Fun \( e \) is \( \lambda x. e' \), which is a value, so we are done.

T-App \( e \) is \( e_1 \) \( e_2 \).

By inversion, \( \cdot \vdash e_1 : \tau' \rightarrow \tau \) and \( \cdot \vdash e_2 : \tau' \) for some \( \tau' \).

If \( e_1 \) is not a value, then \( \cdot \vdash e_1 : \tau' \rightarrow \tau \) and the induction hypothesis ensures \( e_1 \rightarrow e'_1 \) for some \( e'_1 \). Therefore, by E-App1, \( e_1 \) \( e_2 \rightarrow e'_1 \) \( e_2 \).

Else \( e_1 \) is a value. If \( e_2 \) is not a value, then \( \cdot \vdash e_2 : \tau' \) and our induction hypothesis ensures \( e_2 \rightarrow e'_2 \) for some \( e'_2 \). Therefore, by E-App2, \( e_1 \) \( e_2 \rightarrow e_1 \) \( e'_2 \).

Else \( e_1 \) and \( e_2 \) are values. Then \( \cdot \vdash e_1 : \tau' \rightarrow \tau \) and the Canonical Forms Lemma ensures \( e_1 \) is some \( \lambda x. e' \). And \( (\lambda x. e') \) \( e_2 \rightarrow e'[e_2/x] \) by E-Apply, so \( e_1 \) \( e_2 \) can take a step.
We will need the following lemma for our proof of Preservation, below. Actually, in the proof of Preservation, we need only a Substitution Lemma where Γ is ·, but proving the Substitution Lemma itself requires the stronger induction hypothesis using any Γ.

**Lemma (Substitution).** If Γ, x:τ′ ⊢ e : τ and Γ ⊢ e′ : τ′, then Γ ⊢ e[e'/x] : τ.

To prove this lemma, we will need the following two technical lemmas, which we will assume without proof (they’re not that difficult).

**Lemma (Weakening).** If Γ ⊢ e : τ and x ∉ Dom(Γ), then Γ, x:τ′ ⊢ e : τ.

**Lemma (Exchange).** If Γ, x:τ, y:τ2 ⊢ e : τ and y ≠ x, then Γ, y:τ2, x:τ1 ⊢ e : τ.

Now we prove Substitution.

Substitution. The proof is by induction on the derivation of Γ, x:τ′ ⊢ e : τ. There are four cases. In all cases, we know Γ ⊢ e′ : τ′ by assumption.

**T-Const** e is c, so e[e'/x] is c. By **T-Const**, Γ ⊢ c : int.

**T-Var** e is y and Γ, x:τ′ ⊢ y : τ.

If y ≠ x, then y[e'/x] is y. By inversion on the typing rule, we know that (Γ, x:τ′)(y) = τ. Since y ≠ x, we know that Γ(y) = τ. So by **T-Var**, Γ ⊢ y : τ.

If y = x, then y[e'/x] is e′. Γ, x:τ′ ⊢ x : τ, so by inversion, (Γ, x:τ′)(x) = τ, so τ = τ′. We know Γ ⊢ e′ : τ′, which is exactly what we need.

**T-App** e is e1 e2, so e[e'/x] is (e1[e'/x]) (e2[e'/x]).

We know Γ, x:τ′ ⊢ e1 e2 : τ1, so, by inversion on the typing rule, we know Γ, x:τ′ ⊢ e1 : τ2 → τ1 and Γ, x:τ′ ⊢ e2 : τ2 for some τ2.

Therefore, by induction, Γ ⊢ e1[e'/x] : τ2 → τ1 and Γ ⊢ e2[e'/x] : τ2.

Given these, **T-App** lets us derive Γ ⊢ (e1[e'/x]) (e2[e'/x]) : τ1.

So by the definition of substitution Γ ⊢ (e1 e2)[e'/x] : τ1.

**T-Fun** e is λy. e_b, so e[e'/x] is λy. (e_b[e'/x]).

We can α-convert λy. e_b to ensure y ∉ Dom(Γ) and y ≠ x.

We know Γ, x:τ′ ⊢ λy. e_b : τ1 → τ2, so, by inversion on the typing rule, we know Γ, x:τ′, y:τ1 ⊢ e_b : τ2.

By Exchange, we know that Γ, y:τ1, x:τ′ ⊢ e_b : τ2.

By Weakening, we know that Γ, y:τ1 ⊢ e′ : τ′.

We have rearranged the two typing judgments so that our induction hypothesis applies (using Γ, y:τ1 for the typing context called Γ in the statement of the lemma), so, by induction, Γ, y:τ1 ⊢ e_b[e'/x] : τ2.

Given this, **T-Fun** lets us derive Γ ⊢ λy. e_b[e'/x] : τ1 → τ2.

So by the definition of substitution, Γ ⊢ (λy. e_b)[e'/x] : τ1 → τ2.

3
Theorem (Preservation). If \( \cdot \vdash e : \tau \) and \( e \rightarrow e' \), then \( \cdot \vdash e' : \tau \).

Preservation. The proof is by induction on the derivation of \( \cdot \vdash e : \tau \). There are four cases.

T-Const \( e \) is \( c \). This case is impossible, as there is no \( e' \) such that \( c \rightarrow e' \).

T-Var \( e \) is \( x \). This case is impossible, as \( x \) cannot be typechecked under the empty context.

T-Fun \( e \) is \( \lambda x. e_b \). This case is impossible, as there is no \( e' \) such that \( \lambda x. e_b \rightarrow e' \).

T-App \( e \) is \( e_1 e_2 \), so \( \cdot \vdash e_1 e_2 : \tau \).

By inversion on the typing rule, \( \cdot \vdash e_1 : \tau_2 \rightarrow \tau \) and \( \cdot \vdash e_2 : \tau_2 \) for some \( \tau_2 \).

There are three possible rules for deriving \( e_1 e_2 \rightarrow e' \).

E-App1 Then \( e' = e'_1 e_2 \) and \( e_1 \rightarrow e'_1 \).

By \( \cdot \vdash e_1 : \tau_2 \rightarrow \tau \), \( e_1 \rightarrow e'_1 \), and induction, \( \cdot \vdash e'_1 : \tau_2 \rightarrow \tau \).

Using this and \( \cdot \vdash e_2 : \tau_2 \), T-App lets us derive \( \cdot \vdash e'_1 e_2 : \tau \).

E-App2 Then \( e' = e_1 e'_2 \) and \( e_2 \rightarrow e'_2 \).

By \( \cdot \vdash e_2 : \tau_2 \), \( e_2 \rightarrow e'_2 \), and induction \( \cdot \vdash e'_2 : \tau_2 \).

Using this and \( \cdot \vdash e_1 : \tau_2 \rightarrow \tau \), T-App lets us derive \( \cdot \vdash e_1 e'_2 : \tau \).

E-Apply Then \( e_1 \) is \( \lambda x. e_b \) for some \( x \) and \( e_b \), and \( e' = e_b[e_2/x] \).

By inversion of the typing of \( \cdot \vdash e_1 : \tau_2 \rightarrow \tau \), we have \( \cdot, x : \tau_2 \vdash e_b : \tau \).

This and \( \cdot \vdash e_2 : \tau_2 \) lets us use the Substitution Lemma to conclude \( \cdot \vdash e_b[e_2/x] : \tau \).